



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Advanced Incident Response, Threat Hunting, and Digital Forensics (Forensics
at <http://www.giac.org/registration/gcfa>

UNSPOKEN TRUTHS – FORENSIC ANALYSIS OF AN UNKNOWN BINARY (GIAC v1.4)

Louie Velocci, CA, CISA, CISSP, CAA
GCFA Certification

Table of Contents

<u>Background:</u>	4
Figure 1.1 MD5 comparison of unknown binary to SANS provided MD5 hash total	4
<u>Media Preparation:</u>	4
Figure 1.2 Method of Sanitizing Media for Forensic Analysis	4
Figure 1.3 Mounting the USB drive in Linux	5
<u>Setting the Stage – The Known Information:</u>	5
Figure 1.4 File Ownership and Zipinfo commands against unknown Binary	5
<u>Extraction of Unknown Binary:</u>	8
Figure 1.5 Unknown Binary - Unzip command	8
Figure 1.6 Unzipped – Unknown Binary File Permissions	9
<u>Integrity of the Unknown Binary:</u>	9
Figure 1.7 MD5 Comparison of Unknown Binary Gzip	9
Figure 1.8 Decompress Unknown Binary and ‘File’ Output	10
<u>Forensic Analysis Process:</u>	10
Figure 1.9 Mounting dd Image in Linux VM Ware	10
Figure 1.10 Identified Unknown Binary ‘Prog’	11
<u>Is it the Real Thing??:</u>	11
Figure 1.11 Unknown Binary MD5 Comparison and Match	11
<u>Autopsy Image Analysis:</u>	11
Figure 1.12 SleuthKit – Autopsy Forensic Analysis Tool – Screenshot	12
Figure 1.13 Initial Case Setup within Autopsy	12
Figure 1.14 – Autopsy – File Activity / Time Line Creation Process	13
Figure 1.15 Unknown Binary Identification within Autopsy	13
Figure 1.16 Unknown Binary Extraction within Autopsy	14
<u>File MAC/Timeline Analysis:</u>	14
<u>Netcat Reference Explored:</u>	15
Figure 1.17 MD5 Comparison of NetCat Binary found and downloaded	16
Figure 1.18 Interesting References within SANS to Vmware session.	16
Figure 1.19 Unknown RPM References Explored in Google Search	18
<u>Correspondence Found:</u>	18
Figure 1.20 Extracted Communications Relating to Investigation	18
<u>Graphic Images Found:</u>	19
Figure 1.21 Graphic Images retrieved from SANS Unknown Binary Assignment – Physical Media	19
Figure 1.22 Graphic Image retrieved from SANS Unknown Binary Assignment - Ebay	19
<u>Binary Analysis:</u>	20
Figure 1.23 Identified Binary File Permissions	20
Figure 1.24 Modifying Permissions of Unknown Binary to allow Execution	20
Figure 1.25 File Utility Output Against Unknown Binary	20
Figure 1.26 Identification of Linked Libraries	21
<u>Helping Myself to Help:</u>	21
Figure 1.27 Invoking the help function of the Unknown Binary	21
<u>Strings Search:</u>	22
<u>Use of BinText:</u>	22
Figure 1.28 Identification of Unknown Binary Website Reference	23
<u>BMAP Version 1.0.20</u>	23
<u>MD5Sum hash total:</u>	23
Figure 1.29 Bmap and Unknown Binary MD5Sum Comparisons.	23
<u>Libraries Required:</u>	24
Figure 1.30 Identification of Bmap Required Libraries	24

<u>Figure 1.31 Autopsy Existence of Bmap Required Libraries</u>	24
<u>Figure 1.31 Compilation of Bmap and Files Created</u>	24
<u>Footprint Analysis:</u>	25
<u>Do the individual binaries outputs compare?:</u>	25
<u>Figure 1.32 Comparison of Bmap and Unknown Binary Operations</u>	26
<u>Comparison of Help Files</u>	26
<u>Figure 1.34 Extract of Bmap Help File</u>	27
<u>Figure 1.33 Strace Utility of Bmap</u>	29
<u>Potential Questions:</u>	29
<u>Appendix A: Strings output for unknown binary 'prog'</u>	31
<u>Appendix B: Forensic Footprint of Bmap v1.0.20</u>	32
<u>Appendix C: Forensic Equipment Configurations:</u>	34
<u>Appendix D: References Used (in order of appearance)</u>	36

Background:

The first component of this paper outlines my forensic analysis of an unknown binary that is contained within a zip file provided by SANS¹ (http://www.giac.org/gcfa/binary_v1_4.zip), as such I have decided to take some precautionary steps for the first stages of the analysis. To ensure that the unknown binary will not affect a whole system (i.e. corrupt the machine), I will be using a Linux 9.0 VMware session running on one of my forensic laptops.

Since the binary is unknown, I do not want to simply unzip the binary without taking some precautionary steps first. Upon downloading the unknown binary I immediately performed an MD5Sum on the file as received from SANS. This will enable me to ensure the file integrity and completeness once transmitted to the air-gapped forensics box.

Figure 1.1 MD5 comparison of unknown binary to SANS provided MD5 hash total

```
[root@POWERBIRDIE Practical - Binary Analysis]# md5sum fl-160703-jp1.dd.gz
4b680767a2aed974cec5fbcbf84cc97a  fl-160703-jp1.dd.gz
[root@POWERBIRDIE Practical - Binary Analysis]# cat fl-160703-jp1.dd.gz.md5
4b680767a2aed974cec5fbcbf84cc97a  fl-160703-jp1.dd.gz
```

Here I compare the file integrity of the downloaded file from SANS using an MD5sum hash total. The Md5sum hash total of the file downloaded matches that provided by SANS - I am therefore comfortable that no changes have been made during the downloading process. This will become useful if the case were ever to go to a Canadian court that would expect the evidence to meet the 'best evidence' principals referenced in the Canada Evidence Act.

To transfer the files from my networked machine to the air-gapped one, I am using a USB Drive™ 128 Mb – USB memory stick with serial # E-D900-00-4989(B). This will also be used to collect evidence of the forensic procedures undertaken, and transfer it to a networked machine for report writing and retention.

Media Preparation:

As a standard practice before beginning any forensic analysis², to ensure that the media is cleansed from previous evidence; I used the 'dd' utility within Linux to write the complete contents of the USB device with a string of zeros. In theory by dd'ing a device with an image source of /dev/zero the forensic analyst is copying an unlimited source of 'zeros' to the destination media.

Figure 1.2 Method of Sanitizing Media for Forensic Analysis

```
[root@POWERBIRDIE /]# dd if=/dev/zero of=/dev/sda1
```

using a if=/dev/zero - results in a theoretically endless amount of zeros to be written to the device using 'dd'

using a of=/dev/sda1 - results in the USB drive to be the destination for these theoretically endless supply of zeros from the dd process

¹ See Appendix D for more details

² I ensure that the media used for any forensic analysis is cleansed prior to use – as this step ensures that the evidence is admissible and in keeping with the 'best evidence' requirements within the Canadian Evidence Act. (<http://laws.justice.gc.ca/en/C-5/>)

Louie Velocci CA, CISA, CISSP, CAA

If this analysis were being done on a Windows machine, an additional low level format prior to using a ported version of 'dd' from a command line would be appropriate (if an investigator has a machine with EnCase installed this step can be completed using the 'wipe drive' option). While not required it is an old habit which provides an additional level of comfort that no residual evidence is present.

As I am using my Linux forensic machine (see **Appendix C** for a complete description of the equipment used.) for this analysis, I will mount the USB device (`/dev/sda1`) as mount point `/mnt/usb/` using the following command:

Figure 1.3 Mounting the USB drive in Linux

```
[root@POWERBIRDIE root]# mount
/dev/sda2 on / type ext3 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda1 on /boot type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
[root@POWERBIRDIE root]# mount /dev/sda1 /mnt/usb
[root@POWERBIRDIE root]# mount
/dev/sda2 on / type ext3 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda1 on /boot type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
/dev/sda1 on /mnt/usb type ext3 (rw)
[root@POWERBIRDIE root]#
```

The diagram shows the terminal session with annotations. The first two lines of the session are highlighted with a grey box and labeled "Initial mount command - no /mnt/usb present". The command `mount /dev/sda1 /mnt/usb` is highlighted with a red box and labeled "command to mount the usb device". The final line of the session, showing the mounted state, is highlighted with a grey box and labeled "2nd mount command - note /dev/sda1 now mounted as /mnt/usb".

Setting the Stage – The Known Information:

Initially I have used the zipinfo utility with the `-v` flag. This utility provides us with a significant amount of data about contained within the file `binary_v1_4.zip` provided to me by SANS, however does not require me to extract the files. This utility can be very helpful to in gathering important information while preventing the extraction of possible malicious code.

Figure 1.4 File Ownership and Zipinfo commands against unknown Binary

```
[root@POWERBIRDIE SANS]# pwd
/SANS
[root@POWERBIRDIE SANS]# ls -l
total 456
-rwx----- 1 root      root      459502 Apr 19 11:57 binary_v1_4.zip
[root@POWERBIRDIE SANS]# zipinfo -v binary_v1_4.zip
Archive: binary_v1_4.zip  459502 bytes  3 files
```

The diagram highlights specific parts of the terminal session with red arrows and boxes. The path `/SANS` is highlighted with a red arrow and a box labeled "Path of unknown binary on Linux forensic machine". The file listing output is highlighted with a red arrow and a box labeled "directory and file attributes note RX by owner only.". The `zipinfo -v` command and its output are highlighted with a red arrow and a box labeled "Use of zipinfo with the verbose (-v flag) options enabled."

Extract of zipinfo `-v` command: (I choose italicized blue text for any extracted text from the analysis)

Archive: *binary_v1_4.zip* 459502 bytes 3 files

End-of-central-directory record:

*Actual offset of end-of-central-dir record: 459460 (000702C4h)
Expected offset of end-of-central-dir record: 459460 (000702C4h)
(based on the length of the central directory and its expected offset)*

This zipfile constitutes the sole disk of a single-part archive; its central directory contains 3 entries. The central directory is 227 (000000E3h) bytes long, and its (expected) offset in bytes from the beginning of the zipfile is 459233 (000701E1h).

The zipfile comment is 20 bytes long and contains the following text:

===== zipfile comment begins
=====
GCFA binary analysis
===== zipfile comment ends
=====

Central directory entry #1:

f1-160703-jp1.dd.gz

*offset of local header from start of archive: 0 (00000000h) bytes
file system or operating system of origin: Unix
version of encoding software: 2.3
minimum file system compatibility required: MS-DOS, OS/2 or NT
FAT
minimum software version required to extract: 2.0
compression method: deflated
compression sub-type (deflation): normal
file security status: not encrypted
extended local header: no
file last modified on (DOS date/time): 2003 Jul 15 23:03:02
file last modified on (UT extra field modtime): 2003 Jul 16 02:03:01
local
file last modified on (UT extra field modtime): 2003 Jul 16 05:03:01
UTC
32-bit CRC value (hex): 037deebe
compressed size: 458937 bytes
uncompressed size: 474162 bytes
length of filename: 19 characters
length of extra field: 13 bytes
length of file comment: 0 characters
disk number on which file begins: disk 1
apparent file type: binary
Unix file attributes (100400 octal): -r-----
MS-DOS file attributes (01 hex): read-only*

The central-directory extra field contains:

- A subfield with ID 0x5455 (universal time) and 5 data bytes.
The local extra field has UTC/GMT modification/access times.
- A subfield with ID 0x7855 (Unix UID/GID) and 0 data bytes.

There is no file comment.

Central directory entry #2:

f1-160703-jp1.dd.gz.MD5

```
offset of local header from start of archive: 459007 (000700FFh)
bytes
file system or operating system of origin: Unix
version of encoding software: 2.3
minimum file system compatibility required: MS-DOS, OS/2 or NT
FAT
minimum software version required to extract: 1.0
compression method: none (stored)
file security status: not encrypted
extended local header: no
file last modified on (DOS date/time): 2003 Jul 16 00:15:00
file last modified on (UT extra field modtime): 2003 Jul 16 03:14:59
local
file last modified on (UT extra field modtime): 2003 Jul 16 06:14:59
UTC
32-bit CRC value (hex): 75457d32
compressed size: 54 bytes
uncompressed size: 54 bytes
length of filename: 23 characters
length of extra field: 13 bytes
length of file comment: 0 characters
disk number on which file begins: disk 1
apparent file type: text
Unix file attributes (100644 octal): -rw-r--r--
MS-DOS file attributes (00 hex): none
```

The central-directory extra field contains:

- A subfield with ID 0x5455 (universal time) and 5 data bytes.
- The local extra field has UTC/GMT modification/access times.
- A subfield with ID 0x7855 (Unix UID/GID) and 0 data bytes.

There is no file comment.

Central directory entry #3:

prog.MD5

```
offset of local header from start of archive: 459135 (0007017Fh)
bytes
file system or operating system of origin: Unix
version of encoding software: 2.3
minimum file system compatibility required: MS-DOS, OS/2 or NT
FAT
minimum software version required to extract: 1.0
compression method: none (stored)
file security status: not encrypted
extended local header: no
file last modified on (DOS date/time): 2003 Jul 16 00:14:38
file last modified on (UT extra field modtime): 2003 Jul 16 03:14:38
```

```

local
file last modified on (UT extra field modtime): 2003 Jul 16 06:14:38
UTC
32-bit CRC value (hex): 804cc662
compressed size: 39 bytes
uncompressed size: 39 bytes
length of filename: 8 characters
length of extra field: 13 bytes
length of file comment: 0 characters
disk number on which file begins: disk 1
apparent file type: text
Unix file attributes (100644 octal): -rw-r--r--
MS-DOS file attributes (00 hex): none

```

The central-directory extra field contains:

- A subfield with ID 0x5455 (universal time) and 5 data bytes.
- The local extra field has UTC/GMT modification/access times.
- A subfield with ID 0x7855 (Unix UID/GID) and 0 data bytes.

There is no file comment.

As we can see this utility tells us that the operating system of the machine used to create this archive is a Unix (likely Linux machine – based on other analysis below), the zip file was last modified on July 16, 2003 and the zip contains fl-160703-jp1.dd.gz, fl-160703-jp1.dd.gz.MD5, and prog.MD5. The individual file sizes are: 474162, 54, 39 bytes respectively.

Extraction of Unknown Binary:

The above analysis indicates that the binary appears to have adopted the user (UID) and group (GID) attributes of my machine. Using the ‘unzip’ command with the ‘-X’ flag enabled it might be possible to determine the permissions of the original UID/GID of the files as they were set on the machine prior to archival. This will potentially yield additional information that can be used in piecing together seemingly random case evidence.

Figure 1.5 Unknown Binary - Unzip command

```
[root@POWERBIRDIE SANS]# unzip -X binary_v1_4.zip
Archive: binary_v1_4.zip
GCFA binary analysis
  inflating: fl-160703-jp1.dd.gz
  extracting: fl-160703-jp1.dd.gz.MD5
  extracting: prog.md5
```

using the -X flag will
set the unzip utility
to restore the original
UID/GID information for
this archive

Next the following command ‘ls -al’ issued to determine the file permissions and timer at archival.

Figure 1.6 Unzipped – Unknown Binary File Permissions

```
[root@POWERBIRDIE SANS]# ls -al
total 940
drwxr-xr-x  2 root    root        4096 Jan  5 08:55 .
drwxr-xr-x  25 root   root        4096 Jan  5 08:55 ..
-rwx-----  1 root    root      459502 Jan  5 08:56 binary_v1_4.zip
-r-----  1 root    root     474162 Jul 16 02:03 fl-160703-jp1.dd.gz
-rw-r--r--  1 root    root            54 Jul 16 03:14 fl-160703-jp1.dd.gz.md5
```

using 'ls -al' I am able to list the directories content with all contents (-a) and long listing format (l)

Note original archival times/dates

binary_v1_4.zip

fl-160703-jp1.dd.gz

fl-160703-jp1.dd.gz.md5

It is interesting to note that the original machine appears to be using a Unix based kernel (based on UID/GID and directory permissions. As a result my expectation is that this unknown binary will be a Unix based binary as opposed to a Windows one. Additionally I noted that the date on the system used for archival was July 16, 2003. This may be important later on as it may help us validate the existence of the binary at that point in time in the general Internet population.

Since I passed the ‘-X’ flag to the unzip command my expectation was that the utility would have restored the UID/GID of the original system. Given that the UID/GID are still set to root/root respectively then either the system was a Unix system or the user was logged in as root in a root owned directory when tar’d. It is possible that the file was possibly archived on another system environment - Windows perhaps.

Integrity of the Unknown Binary:

Before I start any analysis on this binary, I will confirm the file integrity using the MD5 hash totals provided by SANS. Since the MD5Sum totals are the same I can state that the binary was not altered between the archival process by SANS, and me extracting it in the forensic lab.

Figure 1.7 MD5 Comparison of Unknown Binary Gzip

```
[root@POWERBIRDIE SANS]# cat fl-160703-jp1.dd.gz.md5 prog.md5
4b680767a2aed974cec5fbcbf84cc97a fl-160703-jp1.dd.gz
7b80d9aff486c6aa6aa3efa63cc56880 prog
[root@POWERBIRDIE SANS]# md5sum fl-160703-jp1.dd.gz
4b680767a2aed974cec5fbcbf84cc97a fl-160703-jp1.dd.gz
```

I noted that the MD5sum hash totals are the same - so we can be certain that the fl-160703-jp1.dd file in the SANS archive is the same as the extracted one.

this command extracts the contents of the SANS provided Md5 hash total files.

I extract the contents of the fl-160703-jp1.dd.gz file with the gzip -d command. The resultant file called fl-160703-jp1.dd appears to be an image created with the Unix ‘dd’ utility. We confirm this with the following:

Figure 1.8 Decompress Unknown Binary and ‘File’ Output

```
[root@POWERBIRDIE SANS]# gzip -d fl-160703-jp1.dd.gz
[root@POWERBIRDIE SANS]# ls
binary_v1_4.zip fl-160703-jp1.dd fl-160703-jp1.dd
[root@POWERBIRDIE SANS]# file fl-160703-jp1.dd
fl-160703-jp1.dd: Linux rev 1.0 ext2 filesystem data
```

gzip -d extracts the contents of a gzip file to the current directory.

the 'file' utility tries to identify the file contents based on the 'magic' parameters embedded in the file. These 'magic' files provide a fingerprint of the file type.

Forensic Analysis Process:

Since the file is a 'dd' image I have two options to start the analysis; first I can use Autopsy (<http://www.sleuthkit.org/autopsy/index.php>), or secondly I can mount the image directly in Linux with the command outlined in Figure 1.9. Initially I chose the second option to mount the image directly, however I will also be using Autopsy to perform further do some analysis. I chose this path, as I do not know what to expect, and if the image is mounted properly then no harm will come from the direct mounting.

Figure 1.9 Mounting dd Image in Linux VM Ware

```
[root@POWERBIRDIE SANS]# mount -o loop,ro,noatime,nodev,noexec fl-160703-jp1.dd /mnt/sans-unknown
binary/
[root@POWERBIRDIE SANS]# mount
/dev/sda2 on / type ext3 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sdal on /boot type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
/dev/sdal on /mnt/usb type ext3 (rw)
/SANS/fl-160703-jp1.dd on /mnt/sans-unknownbinary type ext2 (ro,noexec,nodev,noatime,loop=/dev/loop0)
[root@POWERBIRDIE SANS]#
```

Mounting the fl-160703-jp1.dd image with the following flags enabled:
`-o :`
`loop: use the loopback device`
`ro: read only - doesn't impact file integrity`
`noatime: don't adjust last access time (since read only - don't need to specify not to change inode times for modifications / creations - none should be present)`
`nodev: basically asks the system to ignore any devices (i.e. /dev/sda1) within the image.`
`noexec: don't allow binaries to execute - could be very dangerous given a potentially 'compromised machine' with unknown binaries`

shows the image mounted on my machine as /mnt/sans-unknownbinary with the same attributes as specified above.

When mounting the image I wanted to ensure that the integrity of the evidence was preserved. By using the following mount options the data integrity will be preserved:

- loop** – use loopback device
- ro** – read only so no image changes are allowed
- noatime** – don't change last access time
- nodev** – Bypass system devices in image
- noexec** – disable binary execution

Once mounted I change to that directory and perform an 'ls' command to list the content of the image and get the following interesting information, namely the '**prog**' file that we have been told to isolate and perform the analysis on as the unknown binary. Typically a forensic analyst doesn't know the exact items they are looking for so this option would be less effective in a real forensic analysis.

Figure 1.10 Identified Unknown Binary 'Prog'

```
[root@POWERBIRDIE SANS]# cd /mnt/sans-unknownbinary/
[root@POWERBIRDIE sans-unknownbinary]# ls
Docs  John  lost+found  May03  nc-1.10-16.i386.rpm..rpm  prog
[root@POWERBIRDIE sans-unknownbinary]#
```

Is it the Real Thing??:

A quick comparison of the MD5Sum hash totals indicates that this is in-fact the unknown binary.

Figure 1.11 Unknown Binary MD5 Comparison and Match

```
[root@POWERBIRDIE SANS]# cat prog.md5
7b80d9aff486c6aa6aa3efa63cc56880  prog
[root@POWERBIRDIE SANS]# md5sum /mnt/sans-unknownbinary/prog
7b80d9aff486c6aa6aa3efa63cc56880  /mnt/sans-unknownbinary/prog
```

SANS provided Md5Sum of the unknown binary

matching Md5sum hash for the found binary - they are the same thing.

So we have the ‘what’ component of the five ‘W’ questions and at a first glance it appears that we do not have to do anything for me to have been able to extract the unknown binary. As a forensic investigator however, I want to examine all the evidence before making any conclusions. I notice the lost+found directory, and the **nc-1.10-16.i386.rpm** file (this is for a network utility called ‘**netcat**’) were also noted in the ‘ls’ output, this might assist / help us to better understand the ‘why’ and ‘where’ of any inappropriate activity.

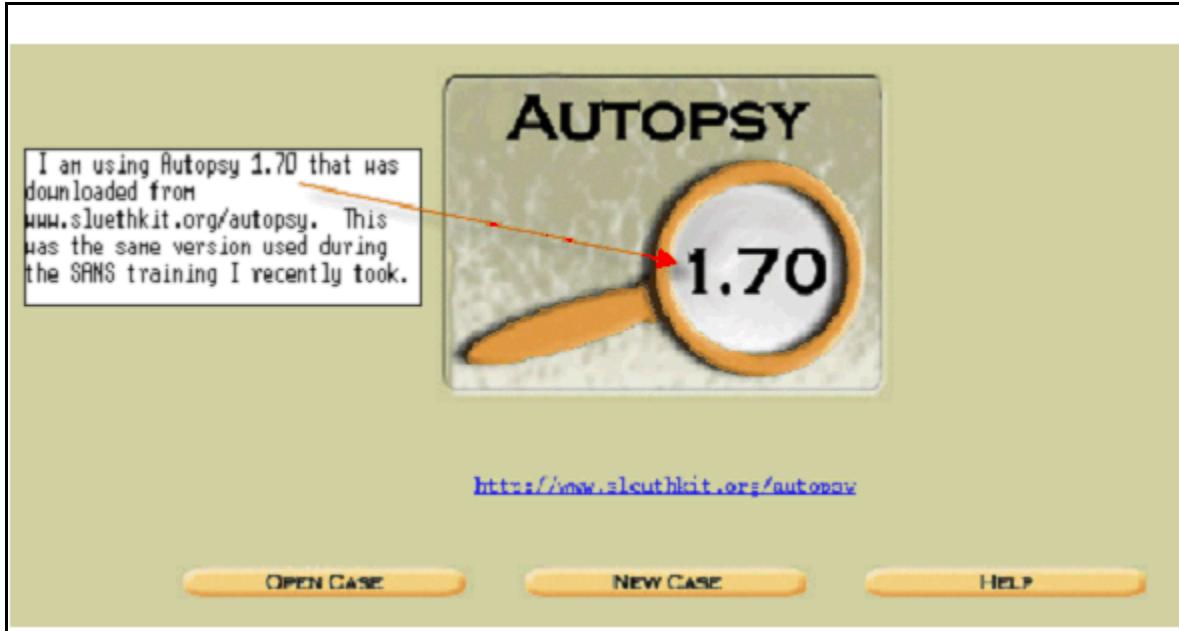
Before going any further I want to load this image into Autopsy and create a timeline as well as other items that I think will help with a thorough investigation.

Autopsy Image Analysis:

To help with the additional analysis of this unknown binary I wanted to perform some additional research in Autopsy³ (version 1.70). Autopsy is an all-encompassing image analysis tool that draws upon several open source utilities; however the individual tools could also be run independently. In this analysis I choose the New Case icon at the bottom and put in the appropriate investigators name, image type and the source location for the image, as well as the MD5hash total database(s). Autopsy takes this information and starts to assess the timeline (modified, accessed and changed).

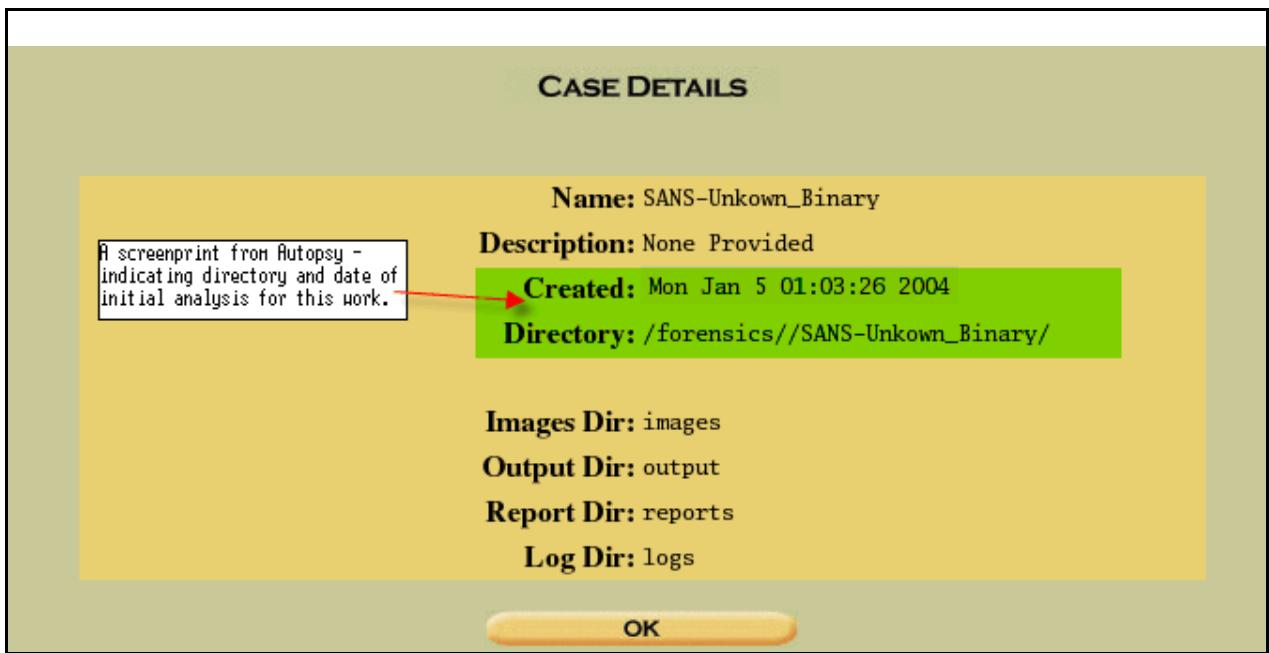
Figure 1.12 SleuthKit – Autopsy Forensic Analysis Tool – Screenshot

³ Autopsy is located at <http://www.sleuthkit.org/autopsy/>



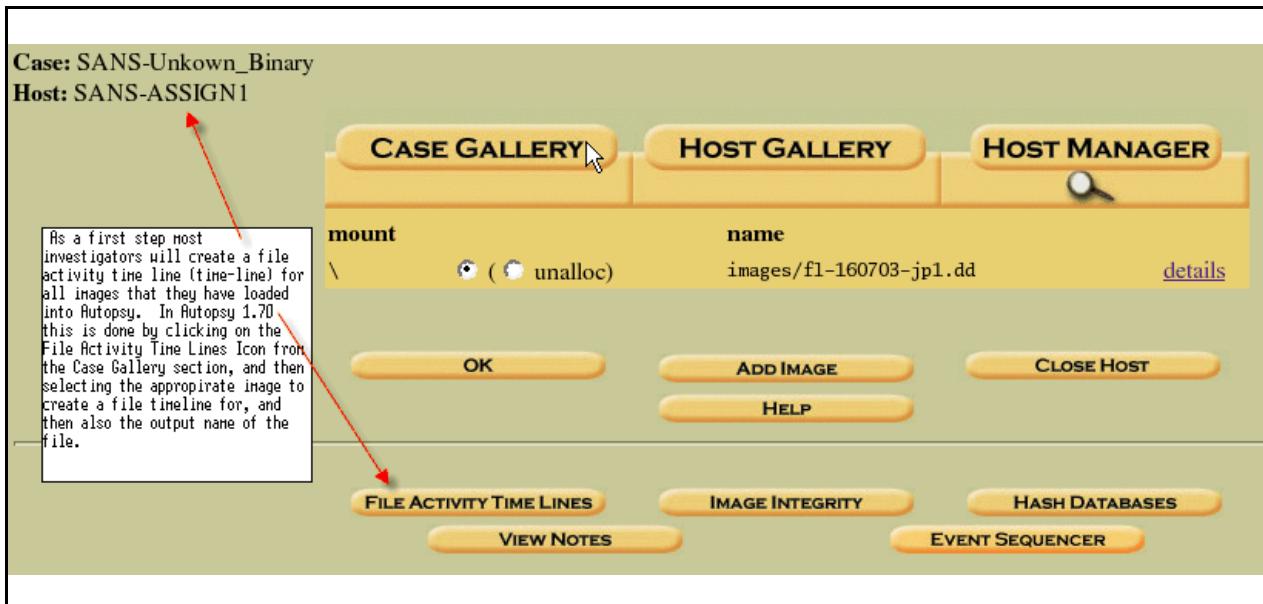
Here are the initial case facts captured in Autopsy:

Figure 1.13 Initial Case Setup within Autopsy



I have compiled the NSRL known MD5hash total databases on my Linux forensic machine. Once in Autopsy I quickly perform a timeline capture (see below) of the image, and then perform a ‘file analysis’ search from within the case gallery sub-menu.

Figure 1.14 – Autopsy – File Activity / Time Line Creation Process



I noted that the image has both allocated and unallocated content which means we will likely have to analyze deleted files during this analysis.

Figure 1.15 Unknown Binary Identification within Autopsy

FILE ANALYSIS **KEYWORD SEARCH** **FILE TYPE** **IMAGE DETAILS** **META DATA** **DATA UNIT** **HELP** **CLOSE**

View Directory: \

DEL	Type	Name	MODIFIED	ACCESSED	CHANGED	SIZE	UID	GID
	dir / in	.. /	2003.07.16 03:03:13 (CDT)	2003.07.16 03:12:39 (CDT)	2003.07.16 03:03:13 (CDT)	1024	0	0
	d / d	.. /	2003.07.16 03:03:13 (CDT)	2003.07.16 03:12:39 (CDT)	2003.07.16 03:03:13 (CDT)	1024	0	0
	r / r	.~5456g.tmp	2003.07.14 11:13:52 (CDT)	2003.07.16 03:11:36 (CDT)	2003.07.14 11:13:52 (CDT)	2592	0	0
	d / d	Docs /	2003.07.14 11:22:36 (CDT)	2003.07.16 03:10:01 (CDT)	2003.07.14 11:43:44 (CDT)	1024	502	502
	d / d	John /	2003.02.03 07:08:00 (ATL)	2003.07.16 03:09:35 (CDT)	2003.07.14 11:49:25 (CDT)	1024	502	502
	d / d	lost+found /	2003.07.14 11:08:09 (CDT)	2003.07.16 03:06:15 (CDT)	2003.07.14 11:08:09 (CDT)	12288	0	0
	d / d	May03 /	2003.05.03 07:10:00 (CDT)	2003.07.16 03:09:49 (CDT)	2003.07.14 11:50:15 (CDT)	1024	502	502
	r / r	nc-1.10-16.i386.rpm..rpm	2003.07.14 11:12:02 (CDT)	2003.07.14 11:12:02 (CDT)	2003.07.14 11:43:57 (CDT)	56950	502	502
	r / r	prog	2003.07.14 11:24:00 (CDT)	2003.07.16 03:12:45 (CDT)	2003.07.16 03:05:33 (CDT)	487476	502	502

Here is the initial output of the file analysis - as can be seen the unknown binary 'prog' is easily visible and by clicking on the title of it, I was able to extract the contents to a file.

Using file analysis it is easy to find the 'prog' file. Once identified it is possible to recover and extract the files using the 'export contents' functionality. Figure 1.16 shows how I identified and exported the unknown binary for further analysis.

Figure 1.16 Unknown Binary Extraction within Autopsy

The screenshot shows the Autopsy Forensic Browser interface. The top menu bar includes FILE ANALYSIS, KEYWORD SEARCH, FILE TYPE, IMAGE DETAILS, META DATA, DATA UNIT, HELP, and CLOSE. Below the menu, a toolbar has buttons for OK, ALL DELETED FILES, and EXPAND DIRECTORIES. A sidebar on the left labeled 'View Directory:' shows a tree view with a single node '\'. The main pane displays two files: 'nc-1.10-16.i386.rpm' and 'prog'. The 'nc-1.10-16.i386.rpm' file is selected, showing its details: created on 2003.07.14 at 11:12:02 (CDT), modified on 2003.07.14 at 11:12:02 (CDT), and last accessed on 2003.07.14 at 11:43:57 (CDT). It has a size of 56950 bytes, 502 blocks, and 502 inodes. The 'prog' file is also listed with similar details. Below the file list, a message from the magic file system indicates it couldn't find any magic files. The bottom pane shows the file's contents as raw hex data.

File MAC/Timeline Analysis:

A timeline provides extremely useful information to a forensic analyst on system modifications, or in reconstructing a chain of events. The process of creating a timeline within Autopsy is rather mundane, in that it is precipitated by clicking an icon and entering in the start date for the timeline and the output file name. You can specify the ending date for a timeline, however you may miss information if the file system dates were altered.

This same process could be done from a command line using the '**'grave-rober tool'** from The Coroners Toolkit ⁴(TCT) assuming that it is installed on the system by using the following command:

grave-robber -E (the -E flag says to grave-robber to grab all file information – however additional flags can be configured - for a complete listing use **grave-robber • • help**)

Once a timeline has been developed an investigator needs to understand how to interpret the results, particularly the differences between the components of the timeline. The timeline represents three system related events for each individual file, commonly referred to as 'MAC' as outlined below:

Timeline Component	Represents
M – modification time	The last time that a file was written
A – access time	The last time a file was read
C – change time	The last time the inode contents were written.

Using Autopsy I extracted the timeline analysis for this image, and noted that the first entry was January 28,2003 which appears to be the creation and access of two of the three images retrieved. Looking for more information about the timelines for the unknown binary, I extracted only the entries related to 'prog':

```
Mon Jul 14 2003 11:24:00 487476 m.. -/-rwxr-xr-x 502 502 18
\prog

Wed Jul 16 2003 03:05:33 487476 ..c -/-rwxr-xr-x 502 502 18
\prog

Wed Jul 16 2003 03:12:45 487476 .a. -/-rwxr-xr-x 502 502 18
```

⁴ Coroners Toolkit (TCT) can be located <http://www.porcupine.org/forensics/tct.html>

\/prog

We can see that the unknown binary was brought onto the machine **July 14th at 11:24**, is **487476** bytes in size and has both a user and group ownership value of 502 (unfortunately we do not have a copy of the /etc/group file to further analyze the memberships of this group (but it is likely all users), in any event the file is set to '**rwxr-xr-x**' meaning that the owning UID/GID have read, write and execute permissions on this file, while all system users have the ability to read and execute this file but not necessarily write to the file. It is interesting to note that the last time the program was accessed was **July 16, 2003 at 3:12 AM**, the same day that the image was taken (which we can assert was the day the evidence was seized).

So far the analysis has shown several references to **nc-1.10-16.i386.rpm** and within the timelines we see specific entries about this nc rpm including the original creation and access date and time of July 14, 2003 at 11:12 AM and a change entry on the same date but at 11:43 AM.

```
\/nc-1.10-16.i386.rpm..rpm  
Mon Jul 14 2003 11:12:15 100430 ma. -rwxr-xr-x 0 0 23  
<fl-160703-jp1.dd-dead-23>
```

We are able to see that Netcat initially appears in this timeline on July 14, 2003 at 11:12 AM. This is likely the compile date and time of Netcat.

```
Mon Jul 14 2003 11:43:57 56950 ..c -/-rwxr-xr-x 502 502 22  
\/nc-1.10-16.i386.rpm..rpm
```

Netcat Reference Explored:

Using Autopsy I was able to extract the nc-1.10-16.i386 rpm binary, and upon analysis determine that the file extracted from the image was the actual rpm that one would use to install 'netcat' on a system. Using a Google search with criteria of '**nc-1.10-16.i386.rpm**' I found that this specific rpm is the Redhat 8.0 rpm for nc. I found and downloaded a copy of the rpm from the website (<http://rpmfind.rediris.es/rpm2html/redhat-8.0-i386/nc-1.10-16.i386.html>) and performed an MD5Sum on it and the one extracted from the fl-160703-jp1.dd image from SANS. The MD5Sum hash totals matched, and so I now know that the system that John Price was likely using was using Redhat 8.0 as it's base operating system, or that he had some dual boot, or VM configuration of Redhat 8.0. Additionally we know have evidence that Netcat was present on his machine.

Figure 1.17 MD5 Comparison of NetCat Binary found and downloaded

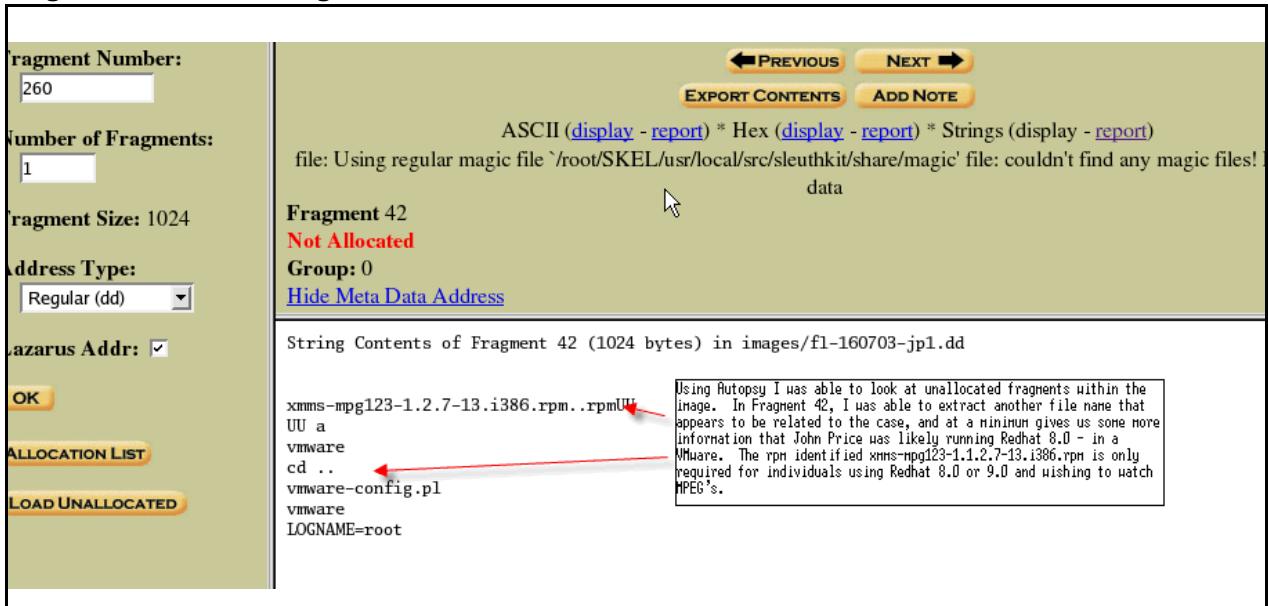
```
[root@POWERBIRDIE SANS]# md5sum nc-1*.rpm  
535003964e861aad97ed28b56fe67720 nc-1.10-16.i386.rpm  
535003964e861aad97ed28b56fe67720 nc-1.10-16.i386.rpm..rpm  
  
The rpm downloaded from the internet (specific  
to Redhat 8.0) - matches the MD5sum of the  
nc-1.10-16.rpm..rpm file extracted from the  
fl-160703-jp1.dd image
```

Netcat (nc) is defined on the same rpm website as 'a simple utility for reading and writing data across network connections, using the TCP or UDP protocols. Netcat is intended to be a reliable back-end tool which can be used directly or easily driven by other programs and scripts. Netcat is also a feature-rich network debugging and exploration tool, since it can create many different connections and has many built-in capabilities'. From experience I know that 'nc' can be used to copy files to / from remote computers, execute commands remotely etc. This leads me to believe that John is likely using

'nc' in connection with his illegal activities and most likely to connect remotely to another machine to retrieve the copyrighted material.

Given that the rpm found was for Redhat 8.0, and the workstation appeared to be a Microsoft based on the presence and use of Microsoft Word 8.0, it is likely that this version of Netcat was either run from a dual boot machine or within a virtual machine. With further investigation of the full image I was able to extract evidence that Netcat was likely run in a virtual machine using Vmware (www.vmware.com).

Figure 1.18 Interesting References within SANS to Vmware session.



Within Autopsy I identified a fragmented inode on the image that contained remnants of references to vmware.

An extract of fragment #42 found using Autopsy

Autopsy string Fragment Report (ver 1.74)

```
-----
Fragment: 42
Length: 1024 bytes
Invalid address in indirect list (too large): 134996352Not allocated to
any meta data structures
MD5 of raw Fragment: e1067497002867b59c8e1953da221c25
MD5 of string output: 5a18a616b1dbb5016819365948c6ac44
Image: /forensics//SANS-Unkown_Binary/SANS-ASSIGN1/images/f1-160703-
jp1.dd
Image Type: linux-ext2
Date Generated: Mon Jan 05 02:05:59 2004
Investigator: unknown
-----
xmms-mpg123-1.2.7-13.i386.rpm..rpmUU
UU a
vmware
cd ..
vmware-config.pl
vmware
LOGNAME=root
```

To ensure that the analysis was complete a Google search with the following search criteria 'xmms-mpg123-1.2.7.13-i386.rpm' confirmed my expectations that this was a Linux source code package. It appears that this

specific rpm is needed by Redhat 8.0 or 9.0 to handle mpeg audio input⁵ – which seems to follow the possible scenario that we are investigating. It also gives further support that we are looking at either a dual boot machine or a machine with VMware installed.

Figure 1.19 Unknown RPM References Explored in Google Search

The screenshot shows a web browser window with the URL <http://havardk.xmms.org/dist/xmms-1.2.7-rh8-rh9-rpm/>. The page content includes:

- The short summary:** You only need to install this rpm to be able to play mp3's in Red Hat 8 / 9: [xmms-mpg123-1.2.7-21.i386.rpm](#)
- Complete set of xmms rpm's for RedHat 8 and 9:** This is a set of RPM's for Red Hat 8 and 9 that includes the mpeg audio input plugin. Red Hat shipped without support for mpeg audio due to concerns about patents. In able to easily be able to add mpeg decoding to a Red Hat install, these RPM's are built with the Red Hat spec file. The RPM's that are distributed by xmms.org are a bit different.
- Source:** [xmms-1.2.7-21.src.rpm](#)
- Binary packages:**
 - [xmms-1.2.7-21.i386.rpm](#)
 - [xmms-mpg123-1.2.7-21.i386.rpm](#) (mpg123 plugin)
 - [xmms-skins-1.2.7-21.i386.rpm](#) (skin package)
 - [xmms-devel-1.2.7-21.i386.rpm](#) (devel package)

A callout box contains the following text:
The rpm identified in the strings output of the unallocated fragment 42 from the f1-160703-jp1.dd image.
Note interesting references to Redhat not shipping with this RPM 'due to concerns about patents'.

Håvard Kvålen <havardk@xmms.org>

In addition to extractable files and various references to rpms', from the image I was able to extract some correspondence between John Price and an individual called Mike.

Correspondence Found:

Figure 1.20 illustrates two Microsoft word documents and several pieces of documentation. Through Autopsy I extracted each of these for further analysis.

Figure 1.20 Extracted Communications Relating to Investigation

The screenshot shows a file extraction interface with the following files listed:

- DVD-Playing-HOWTO-html.tar
- Letter.doc
- MP3-HOWTO-html.tar.gz
- Kernel-HOWTO-html.tar.gz
- Mikemsg.doc
- Sound-HOWTO-html.tar.gz

A callout box contains the following text:
Documentation found within the f1-160703-jp1.dd image from SAMS. Note that several documents reference use of various media file types (DVD, MP3, Sound files).
These might have been useful for John when trying to set up his Linux machine to play various media types - assuming that he was illegally copying DVD's, CD's or Music in some format.
Since we didn't find any actual copied media we would not be able to say with 100 percent certainty.

One word document details a communication between John Price and an individual referred to as 'Mike' (located in a file called Mikemsg.doc, which appears to have been created on July 14th, or 2 days before the image was created (so close to the investigation initiation).

'Hey Mike,
I received the latest batch of files last night and I'm ready to rock-n-roll (ha-ha).

⁵ Taken from <http://havardk.xmms.org/dist/xmms-1.2.7-rh8-rh9-rpm/>

I have some advance orders for the next run. Call me soon.
JP'

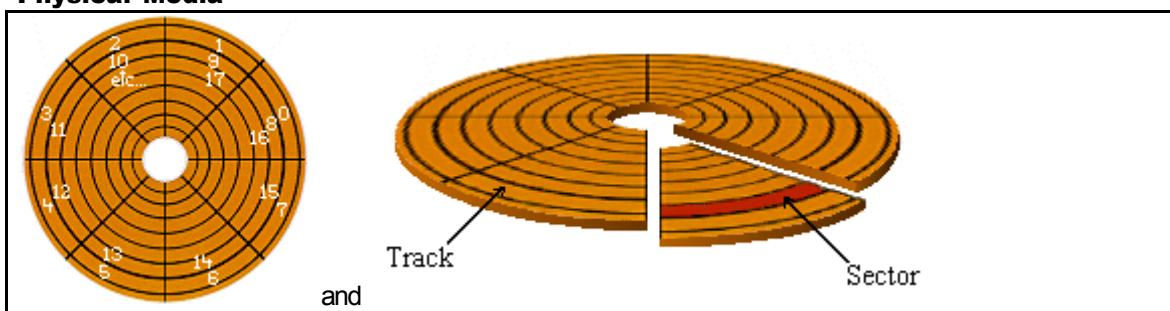
This communication is interesting as it indicates that John Price received ‘the latest’ indicating that there has been past batches of copyrighted material illegally copied and gives a sense that he is not acting alone. Additionally he appears to have a pre-defined market for his stolen goods and is willing to source items that they would like. This would typically indicate that he trusts these individuals (both to buy illegal copies, and to formalize the process enough to start taking requests).

The other items of note was the reference to ‘last night’ and ‘next run’, this might give us a sense that the individual is using the IT resources overnight, or is collecting the files at night and processing them during normal business hours. Additionally when I read ‘next run’ I envision a process which is actually re-copying material into a more useable format (i.e. copying CD’s). Finally it is signed by ‘JP’, which I have taken to be John Price our suspect. This links his activities to him.

Graphic Images Found:

During our analysis I was also able to identify three (3) pictures located within the fl-160703-jp1.dd image. Two of the images sect-num.gif, and sector.gif respectively – seem to depict a hard-disk and the allocations of physical media sectors.

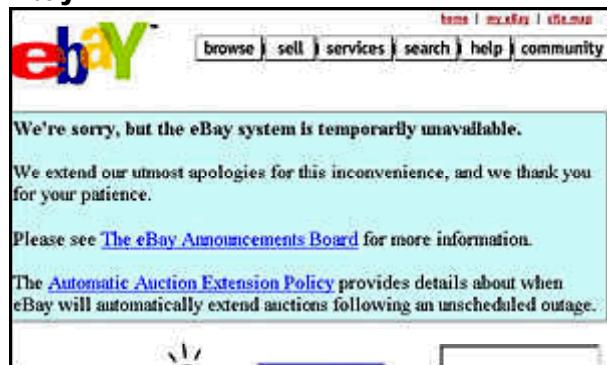
Figure 1.21 Graphic Images retrieved from SANS Unknown Binary Assignment – Physical Media



Since the scenario provided by SANS indicates that John Price was using the company’s IT resources to illegally distribute copyrighted material, this might indicate that John had to do some research on how to make an exact copy (and hence the reason behind track and sectors).

Figure 1.22 depicts the final image ebay300.gif which appears to be un-related to this investigation – although at first I thought it might be a means for John Price to sell the copyrighted materials. I did not investigate this further at this point in time.

Figure 1.22 Graphic Image retrieved from SANS Unknown Binary Assignment - Ebay



So far our analysis tends to lead us to John Price copying some form of entertainment (Music, Video, DVD) using an unknown program. Given the nature of the items he is likely copying, my expectation is that the unknown binary will be a ‘ripper’ or image utility of some type to make illegal copies of DVD’s, CD’s or other media. Such a utility would be used to possibly break the encryption around a movie for instance and then allow the person to make illegal copies of the same CD or DVD. Let’s move to the binary analysis now.

Binary Analysis:

Having identified the file called ‘*prog*’ as the unknown binary from SANS and verifying its integrity I started by using the ‘ls -l *prog*’ command to show the file permissions for *prog*.

Figure 1.23 Identified Binary File Permissions

```
[root@POWERBIRDIE sans-unknownbinary]# ls -l prog  
-rwxr-xr-x    1 JLO        JLO        487476 Jul 14 11:24 prog
```

Note that JLO is one of the UIDs on my forensic machine and so the name is not that of the original machine, rather it has inherited the user and group references that are on this particular machine. What is interesting is that the permissions is executable by all individuals on the system. This would of course assume that we hadn’t set the mount options to ‘read only’ and ‘noexec’. As it is currently mounted we will not be able to execute the binary for our investigation. So I will first copy ‘*prog*’ to a partition that will allow me to execute the binary when necessary. I will at the same time change the permissions to ensure that it isn’t accidentally executed by another system user (which is unlikely as it is an air-gapped laptop). To do this we use ‘chmod 700 *prog*’ and perform another ‘ls -al *prog*’

Figure 1.24 Modifying Permissions of Unknown Binary to allow Execution

```
[root@POWERBIRDIE sans-unknownbinary]# cp ./prog /SANS/  
[root@POWERBIRDIE sans-unknownbinary]# cd /SANS/  
[root@POWERBIRDIE SANS]# chmod 700 prog  
[root@POWERBIRDIE SANS]# ls -al prog  
-rwx----- 1 root      root      487476 Jan  5 09:58 prog
```

Note permissions are changed to reflect that only root can now execute.

Copy ‘*prog*’ to another partition and change to that directory

Change the file permissions so that only root can execute this unknown binary (within a VM so any potential damage would not be permanent).

Comfortable that only root can execute this binary, it is important to extract as much information without actually executing it. To extract some basic information about the binary I will use the ‘file’ utility in Linux to extract any useful information.

Figure 1.25 File Utility Output Against Unknown Binary

```
[root@POWERBIRDIE SANS]# file prog  
prog: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, stripped
```

As Figure 1.25 shows, the binary is a Linux executable that is statically linked and is considered version 1. This leads me to believe that it is not dependant on the existence of any shared libraries. I will confirm this with the ‘ldd’ utility that will identify any interesting shared libraries that might be needed to execute the binary.

Figure 1.26 Identification of Linked Libraries

```
[root@POWERBIRDIE SANS]# ldd prog  
not a dynamic executable
```

So we are not looking for a binary that requires shared libraries to function properly, this might mean that the program could be run as a stand-alone or run from a CD or floppy. As a next step I will see if any help files exist.

Helping Myself to Help:

In an attempt to see the usage pattern of this unknown binary I decided to try to see if there was a help file associated with it. This test is rather harmless as it doesn't actually execute the binary – only shows you the help file.

To test for this I used '`./prog • • help`', which invokes a help file if available. The output was interesting, and so I re-ran the same command but added an optional output redirection flag so that I could review the help file in a text editor, or possibly use it for comparative purposes.

Figure 1.27 Invoking the help function of the Unknown Binary

```
[root@POWERBIRDIE SANS]# ./prog --help  
prog:1.0.20 (07/15/03) newt  
Usage: prog [OPTION]... [<target-filename>]  
use block-list knowledge to perform special operations on files  
  
--doc VALUE  
where VALUE is one of:  
version display version and exit  
help display options and exit  
man generate man page and exit  
sgml generate SGML invocation info  
--mode VALUE  
where VALUE is one of:  
m list sector numbers  
c extract a copy from the raw device  
s display data  
p place data  
w wipe  
chk test (returns 0 if exist)  
sb print number of bytes available  
wipe wipe the file from the raw device  
frag display fragmentation information for the file
```

Interestingly enough we now have a version and date, and a reference to newt. It is likely that newt either created, or is responsible for the upkeep of whatever the binary is.

It appears as if whoever compiled this at SANS did a thorough job of stripping the actual name of the binary - the outputted helpfile will be useful as we can use the 'diff' command against the help file now.

The rest of the help file makes references to various configuration specific help. I decided that since some of the relevant information was obfuscated, that I might yield better results from the use of the strings utility that reads the contents of a file and shows any printable characters.

Strings Search:

Often extremely useful information is contained within the source code of the binaries / programs themselves. It might be for de-bugging purposes, programmer documentation or simply a wise-guy programmer wanting to take some personal credit. In any event it is also very useful in most cases to help identify unknown

binaries. Next I run ‘strings’ against prog and output it to a file. The output was eighty-nine (89) pages in length, so I have extracted the parts that I used from this output and incorporated them into this document. For a full listing of the output file see **Appendix A**.

‘1.0.20 (07/15/03)’ – appeared to be a version and date reference for this file – so potentially useful for comparative purposes later on.

‘newt’ – appears to be a specific references to an individual who was also referenced in the help files – possibly an Internet ‘handle’ for whomever is taking care of bug fixes or at least tracking them.

‘use block-list knowledge to perform special operations on files’ – appeared to be some specific help related information for ‘prog’

Use of BinText:

Since the strings output wasn’t the easiest to use and didn’t provide me with many valuable pieces of information I decided to use **BinText**⁶ to load the ‘prog’ file and extract the following text which appeared to be part of a help file. Specifically I was interested in the following:

‘0004C880 0004C880 0 use block-list knowledge to perform special operations on files’

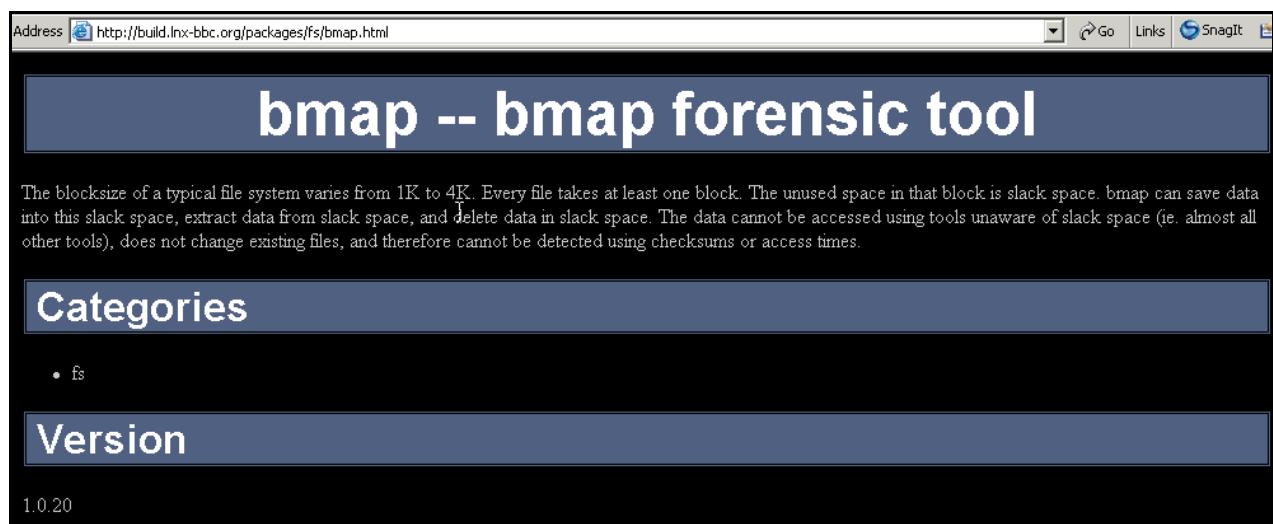
This appeared to be some binary specific type information, which I confirmed by moving to the networked machine and performing a Google⁷ search on the following:

‘use block-list knowledge to perform special operations on files’

One of the Google search pages returned was <http://lwn.net/2000/0420/announce.php3> and referenced a product called bmap v1.0.17. Interested in this I decided to see first if there was a version 1.0.20 (using Google with a search criteria of ‘bmap 1.0.20’) – which in fact there was. I downloaded it from another website (<http://build.lnx-bbc.org/packages/fs/bmap.html>)

Figure 1.28 depicts a likely match to the unknown binary.

Figure 1.28 Identification of Unknown Binary Website Reference



⁶ BinText is a text extractor freeware tool that can be located at www.foundstone.com/resources/proddesc/bintext.htm.

⁷ www.google.ca (Canadian version) www.google.com (U.S. version)
www.foundstone.com/resources/proddesc/bintext.htm

It is interesting that the references and analysis now appears to point to a forensic imaging tool not a ripper program. This would make sense as John would want to make exact copies of media, then he would be using an imaging tool that is capable of performing ‘true’ copies, which typically requires bit-stream-copies⁸.

BMAP Version 1.0.20

To determine if BMAP was in fact the unknown binary, I felt it prudent to download the source from this website, and then perform some analysis on it. The first step was to obtain the source file and un-gzip and un-tar it, in essence I uncompressed the source files and loaded them to my Unix based analysis machine (once downloaded from my networked machine with Internet connectivity).

MD5Sum hash total:

Ideally we would be able to compare the outputs of the MD5Sum hash totals for each file and have a perfect match. Unfortunately this is not typically the way that things work in a forensic investigation and when I tried to compare the MD5Sum hashes of these two files, they were different. This indicates that the content of each file has at least a single character difference.

Figure 1.29 Bmap and Unknown Binary MD5Sum Comparisons.

```
[root@POWERBIRDIE bmap-1.0.20]# md5sum prog bmap
7b80d9aff486c6aa6aa3efa63cc56880  prog
ea6b4ce0c141263dfe9edf75d79b218a  bmap
```

This makes sense as we will see later from the footprint of bmap that it writes specific lines into the files about the compile date, which would result in different MD5's. Additionally any variants in the system configuration, such as the error below that I received during compilation would result in slight variances in the way that bmap was compiled on my system, even if it works fine against our test files.

```
“make: sgml2lateX: Command not found
make: *** [doc] Error 127”
```

Libraries Required:

Having identified that the MD5's do not match, I decided to see which libraries bmap uses and see if I can find evidence of the same dependencies for the unknown binary. To do this I used the ‘ldd’ utility against the bmap executable and identified that it calls **libc.so.6** and **ld-linux.so.2** as illustrated in Figure 1.30 below.

Figure 1.30 Identification of Bmap Required Libraries

```
[root@POWERBIRDIE bmap-1.0.20]# ldd ./bmap
 libc.so.6 => /lib/i686/libc.so.6 (0x42000000)
 /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

The same command against the unknown binary did not result in similar results, as the unknown binary did not have the dependencies available. From the image however, I was able to use Autopsy to extract references for the same libraries using a keyword search on **libc**. While this isn't conclusive evidence on its own it does lead us to further believe that the unknown binary is bmap. Next I wanted to look for any type of unique characteristics that the software might leave during a compile.

Figure 1.31 Autopsy Existence of Bmap Required Libraries

⁸ Bit-Stream refers to a process where the image is made by copying over the ones and zeros of a file one bit at a time – resulting in a true likeness to the original. This is similar to forensic image using the ‘dd’ utility.

<p>New Search</p> <p>3 occurrences of libc\.\.so were found</p> <p>Search Options: Case Insensitive</p> <hr/> <p>Fragment 194 (Hex - Ascii)</p> <p>1: 219 (libc.so.6) 2: 229 (libc.so.6(GL)) 3: 250 (libc.so.6(GL))</p>	<p>file: Using regular magic file '/root/SKEL/usr/local/src/sleuthkit/share/magic' file: couldn't find any magic files! File Type: data</p> <p>Fragment 194</p> <p>String Contents of Fragment 194 (1024 bytes) in images/f1-160703-jp1.dd</p> <pre> root root root root root root nc-1.10-16.src.rpm rpmlib(PayloadFilesHavePrefix) rpmlib(CompressedFileNames) /bin/sh libc.so.6 libc.so.6(GLIBC_2.0) libc.so.6(GLIBC_2.2) </pre>
--	---

Figure 1.31 Compilation of Bmap and Files Created

```
[root@POWERBIRDIE bmap-1.0.20]# ls
bclump          bmap-makefile-output.txt  dev_builder      libbmap.o   slacker
bclump.c        bmap.o                  dev_builder.c    LICENSE     slacker.c
bclump-invoke.sgml bmap.sgml           dev_entries.c   Makefile    slacker-invoke.sgml
bclump.o        bmap.sgml.m4          dev_entries.o   man        slacker-modules.c
bmap            bmap.spec             include          index.html  prog
bmap.c          config.h              libbmap.c       README     slacker.o
bmap-invoke.sgml COPYING             libbmap.c       test.mpg
```

I have copied the unknown binary prog into the bmap-1.0.20 directory. I have also created a file called test.mpg with some text embedded.

Footprint Analysis:

Each piece of software leaves a unique footprint as it is compiled and/or executed on a system. For instance bmap has a unique way of defining echo statements into files as it is compiling. As an example a file within the bmap directory called config.h contains the compile date, 01/09/04, which is the system date on my machine during compilation. We see other examples in the help files and others.

```
[root@POWERBIRDIE bmap-1.0.20]# cat config.h
#ifndef NEWT_CONFIG_H
#define NEWT_CONFIG_H
#define VERSION "1.0.20"
#define BUILD_DATE "01/09/04" ←
#define AUTHOR "newt@scyld.com"
#define BMAP_BOGUS_MAJOR 123
#define BMAP_BOGUS_MINOR 123
#define BMAP_BOGUS_FILENAME "/.../image"
#define _FILE_OFFSET_BITS 64
#endif
```

I noted during the compiling of bmap 1.0.20 that specific, static echo statements are written to a file called config.h within the bmap directory.

Knowing that these unique characteristics exist, an investigator or even an administrator could now target a system and create a simple shell script that would look for a combination of a **config.h** files with the term '**newt@scyld.com**'. Additionally they could look for the mere presence of **newt_conf.h** files on their system.

Do the individual binaries outputs compare?:

One way to be able to prove that the unknown binary is actually bmap v1.0.20 is to see if the two files operate in the same fashion on the same test file. To do this I have compiled bmap 1.0.20 into a VM session, and also copied the unknown binary to the same directory. Additionally I created a file called test.mpg to use in the analysis of both files.

Figure 1.32 Comparison of Bmap and Unknown Binary Operations

```
[root@POWERBIRDIE bmap-1.0.20]# cat sanstest.mpg
3297240
3297241
3297242
3297243
3297244
3297245
3297246
3297247

[root@POWERBIRDIE bmap-1.0.20]# cat sanstest1.mpg
3297240
3297241
3297242
3297243
3297244
3297245
3297246
3297247
```

These numbers refer to the actual blocks that are being copied. Note that both binaries output the same content and operate in the same fashion.

As we can see the two binaries handle the test.mpg file in exactly the same way, and appear to be performing a bit-stream level copying of the files. The files only contain one line of text so the output appears to be an appropriate amount.

Comparison of Help Files

Finally, as a last step before concluding that the unknown binary is bmap 1.0.20 I decided to compare the outputs of the help files that are associated with each. Figure 1.34 shows us the help file from bmap 1.0.20:

Figure 1.34 Extract of Bmap Help File

```
[root@POWERBIRDIE bmap-1.0.20]# ./bmap --help
bmap:1.0.20 (01/09/04) newt@scyld.com
Usage: bmap [OPTION]... [<target-filename>]
use block-list knowledge to perform special operations on files

--doc VALUE
where VALUE is one of:
version display version and exit
help display options and exit
man generate man page and exit
sgml generate SGML invocation info
--mode VALUE
where VALUE is one of:
map list sector numbers
carve extract a copy from the raw device
```

Extract of the help file associated with bmap 1.0.20

Now we examine the help file associated with 'prog' (the unknown binary) as illustrated in Figure 1.35 below.

Figure 1.35 Extract of Unknown Binary Help File

```
[root@POWERBIRDIE bmap-1.0.20]# ./prog --help
prog:1.0.20 (07/15/03) newt
Usage: prog [OPTION]... [<target-filename>]
use block-list knowledge to perform special operations on files

--doc VALUE
where VALUE is one of:
version display version and exit
help display options and exit
man generate man page and exit
sgml generate SGML invocation info
--mode VALUE
where VALUE is one of:
m list sector numbers
```

This is the output of the help file associated with the unknown binary. I noted that while not identical there is little doubt that these two binary files are not variants of each other.

It is also interesting to note that the date of compilation is located within these files. This is consistent with the echo comments in bmap's footprint.

Figure 1.36 Comparison of the Functions of Each Binary

```
[root@POWERBIRDIE bmap-1.0.20]# ./bmap --outfile sanstest.mpg test.mpg
[root@POWERBIRDIE bmap-1.0.20]# ./prog --outfile sanstest1.mpg test.mpg
[root@POWERBIRDIE bmap-1.0.20]# ls -l sans*
-rwxr-xr-x 1 root      root      64 Jan 10 00:27 sanstest1.mpg
-rwxr-xr-x 1 root      root      64 Jan 10 00:27 sanstest.mpg
[root@POWERBIRDIE bmap-1.0.20]# md5sum sans*
7898c0e0cb08eb232bee811f1d3325cb  sanstest1.mpg
7898c0e0cb08eb232bee811f1d3325cb  sanstest.mpg
```

Finally I performed an Md5sum hash total of the output from two different binaries and noted that the hash totals are the same. This could only happen if the binaries acted in the same fashion and wrote the same files.

Running the two independent files against my test.mpg file

Performing an ls -l command for all sans* files. Notice the same file size, permissions and ownership.

All the above analysis leads me to believe that the unknown binary is bmap v1.0.20.

STRACE Comparison:

Another good approach to seeing how a binary interacts with a system is to use the debugging tool ‘strace’, which traces the interactions the binary performs on a system. While I do not expect that the output would be the same, as an investigator it would provide some additional assurance if there were similar characteristics in each file. We start off with the strace output for the unknown binary, as it is likely going to have fewer interactions on the system as it was not compiled on it.

```
[root@POWERBIRDIE bmap-1.0.20]# strace ./prog
execve("./prog", ["./prog"], /* 32 vars */) = 0
fcntl64(0, F_GETFD)                 = 0
fcntl64(1, F_GETFD)                 = 0
fcntl64(2, F_GETFD)                 = 0
uname({sys="Linux", node="POWERBIRDIE", ...}) = 0
geteuid32()                          = 0
getuid32()                           = 0
getegid32()                          = 0
getgid32()                           = 0
brk(0)                               = 0x80bedec
brk(0x80bee0c)                      = 0x80bee0c
brk(0x80bf000)                      = 0x80bf000
brk(0x80c0000)                      = 0x80c0000
write(2, "no filename. try '--help' for he", 36) = 36
exit(2)                             = ?
```

I then ran the same utility on the bmap binary, and note that there is several similarities, and while not conclusive – once again a good indication that we have identified the unknown binary as bmap.

Figure 1.33 Strace Utility of Bmap

Unknown Binary Program Identification:

Based on the above analysis the unknown binary is called **bmap v1.0.20**, and is **487 476** bytes in size.

Having identified the binary as BMAPV1.0.20, at this point I would likely confront John Price with our forensic analysis to try and pose the following questions that might help further understand the full spectrum of evidence.

Potential Questions:

If I were able to sit down with John Price and was given the chance to ask five questions I would likely use the following questions which I feel will be helpful in the investigation but not come across as a direct attack on him.

- 1.) ‘John, during the investigation I have identified several programs (binaries) that are not standard programs for our deployments. Given that this was your machine did you happen to load any non-standard programs on your machine? If you did we would like to know what they were so that we can identify if you possibly installed a ‘trojan’d version of a file? It might be possible that you could have inadvertently loaded files without knowing it?’ [it is important to always allow them the possibility of helping themselves]
 - 2.) ‘John, from what I was able to analyze it appears that somehow the company’s IT resources were being used for non-business purposes during non-business hours Do you think that this is a serious issue?’ [this will give an investigator a sense of the moral fiber of the individual that they are investigating, in practice similar questions have been enough for an individual to state that they were wrong to do this type of thing but didn’t understand the gravity of their actions]
 - 3.) ‘John, the evidence seems to lead me to believe that you and other individuals were using the company’s computers and network resources inappropriately, apparently to try and break copyright laws. Management is taking this very seriously and wants to ensure that this does not happen again. Is there anyone else that was involved?’
 - 4.) ‘John is it possible that I am not getting the full picture based on the evidence that I am seeing, or that it seems worse than it really was? Did you care to help clarify your actions? – I would really like to clear this up before it gets out of

hand' [this gives them the sense that you have found evidence of wrong doing and might infer the worst possible solution – it might be enough for them to want to try and clear their name / of confess to their actions]

- 5.) ‘John, I have to admit that based on the evidence that we were able to collect, and the fact that you attempted to wipe some evidence from the system prior to your departure does not look good for you. Why did you try to wipe information from your computer? What were you afraid we would find? [sometimes you have to play it hard with these individuals, and often their reasoning for destroying evidence is not what we think they did it for.]

Appendix A: Strings output for unknown binary ‘prog’

This is a embedded PDF document that contains the full content of the strings output against the ‘prog’ file.
I choose to embed the document as a PDF because the full file was approximately eighty plus (80+) pages.



"Output of command
strings prog.pdf"

Appendix B: Forensic Footprint of Bmap v1.0.20

This is the output of the ‘make’ command for bmap 1.0.20.

```
echo "#ifndef NEWT_CONFIG_H" > config.h
echo "#define NEWT_CONFIG_H" >> config.h
echo "#define VERSION \"1.0.20\"" >> config.h
echo "#define BUILD_DATE \"01/09/04\"" >> config.h
echo "#define AUTHOR \"newt@scyld.com\"" >> config.h
echo "#define BMAP_BOGUS_MAJOR 123" >> config.h
echo "#define BMAP_BOGUS_MINOR 123" >> config.h
echo "#define BMAP_BOGUS_FILENAME \"/.../image\" >> config.h
echo "#define _FILE_OFFSET_BITS 64" >> config.h
echo "#endif" >> config.h
if [ -n mft ] ; then make -C mft ; fi
make[1]: Entering directory `/UNKNOWN_Binary/bmap-1.0.20/mft'
echo "#define MFT_VERSION \"0.9.2\"" > mft_config.h
echo "#define MFT_BUILD_DATE \"01/09/04\"" >> mft_config.h
echo "#define MFT_AUTHOR \"newt@scyld.com\"" >> mft_config.h
cc -Wall -g -I. -Iinclude -c -o option.o option.c
cc -Wall -g -I. -Iinclude -c -o log.o log.c
log.c:354: warning: `syslog_dispatch' defined but not used
log.c:361: warning: `html_dispatch' defined but not used
cc -Wall -g -I. -Iinclude -c -o helper.o helper.c
ld -r --whole-archive -o libmft.a option.o log.o helper.o
make[1]: Leaving directory `/UNKNOWN_Binary/bmap-1.0.20/mft'
cc -Wall -g -Imft/include -Iinclude -Lmft -lmft dev_builder.c -o
dev_builder
mft/libmft.a: In function `mft_log_perror':
/UNKNOWN_Binary/bmap-1.0.20/mft/log.c:297: `sys_errlist' is deprecated; use
`str
error' or `strerror_r' instead
/UNKNOWN_Binary/bmap-1.0.20/mft/log.c:294: `sys_nerr' is deprecated; use
`strerr
or' or `strerror_r' instead
cc -Wall -g -Imft/include -Iinclude -c -o bmap.o bmap.c
bmap.c: In function `main':
bmap.c:371: warning: implicit declaration of function `dprintf'
cc -Wall -g -Imft/include -Iinclude -c -o libbmap.o libbmap.c
./dev_builder > dev_entries.c
cc -Wall -g -Imft/include -Iinclude -c -o dev_entries.o dev_entries.c
cc -Lmft -lmft bmap.o libbmap.o dev_entries.o -o bmap
mft/libmft.a: In function `mft_log_perror':
/UNKNOWN_Binary/bmap-1.0.20/mft/log.c:297: `sys_errlist' is deprecated; use
`str
error' or `strerror_r' instead
/UNKNOWN_Binary/bmap-1.0.20/mft/log.c:294: `sys_nerr' is deprecated; use
`strerr
or' or `strerror_r' instead
cc -Wall -g -Imft/include -Iinclude -c -o slacker.o slacker.c
cc -Wall -g -Imft/include -Iinclude -c -o slacker-modules.o slacker-modules.c
cc -Lmft -lmft slacker.o slacker-modules.o libbmap.o dev_entries.o -o
slacker
mft/libmft.a: In function `mft_log_perror':
/UNKNOWN_Binary/bmap-1.0.20/mft/log.c:297: `sys_errlist' is deprecated; use
`str
error' or `strerror_r' instead
/UNKNOWN_Binary/bmap-1.0.20/mft/log.c:294: `sys_nerr' is deprecated; use
`strerr
```

```
or' or `strerror_r' instead
cc -Wall -g -Imft/include -Iinclude -c -o bclump.o bclump.c
bclump.c:313: warning: missing braces around initializer
bclump.c:313: warning: (near initialization for `options[1].defval')
cc -Lmft -lmft bclump.o -o bclump
mft/libmft.a: In function `mft_log_perror':
/UNKNOWN_Binary/bmap-1.0.20/mft/log.c:297: `sys_errlist' is deprecated; use
`str
error' or `strerror_r' instead
/UNKNOWN_Binary/bmap-1.0.20/mft/log.c:294: `sys_nerr' is deprecated; use
`strerr
or' or `strerror_r' instead
for i in bmap slacker bclump ; do ./${i} --sgml > ${i}-invoke.sgml ; done
m4 < bmap.sgml.m4 > bmap.sgml
sgml2latex bmap.sgml
make: sgml2latex: Command not found
make: *** [doc] Error 127
```

Appendix C: Forensic Equipment Configurations:

I used two (2) IBM T40 laptops for this analysis – each has some common items (i.e. processors) and then specifics about the loadset and operating system. I have broken this appendix into these two same areas.

Common Characteristics

Processor(s)

Model : Intel(R) Pentium(R) M processor 1500MHz

Speed : 797MHz

L2 On-board Cache : 1024kB ECC synchronous ATC

Mainboard and BIOS

Bus(es) : AGP PCI PCMCIA CardBus USB SMBus/i2c

System BIOS : IBM 1RET36WW (1.07)

Mainboard : IBM 2373EU1

Front Side Bus Speed : 1x 89MHz (89MHz data rate)

Installed Memory : 1023MB

Video System

Monitor/Panel : IBM ThinkPad 1024x768 TFT LCD panel

Adapter : ATI MOBILITY RADEON 7500

Specific Differences Include:

Machine 1 Internal Network Machine //Haliveloclo1⁹

Querying information for HALIVELOCLO1...

System information for \\HALIVELOCLO1:

Uptime:	0 days 3 hours 37 minutes 1 second
Kernel version:	Microsoft Windows 2000, Uniprocessor Free
Product type:	Professional
Product version:	5.0
Service pack:	3
Kernel build number:	2195
Registered organization:	ABC CORP
Registered owner:	ABC CORP
Install date:	03/10/2003, 1:14:58 AM
Activation status:	Not applicable
IE version:	6.0000
System root:	C:\\WINNT
Processors:	1
Processor speed:	1.4 GHz
Processor type:	Intel(R) Pentium(R) M processor
Physical memory:	1022 MB
Video driver:	ATI MOBILITY RADEON 7500

Machine 2 AIRGAPPED Network Machine //POWERBIRDIE

Output from ‘uname -an’ command

Linux POWERBIRDIE 2.4.20-8 #1 Thu Jan 13 17:54:28 EST 2003 i686 i686 i386 GNU/Linux

Logical Storage Devices:

⁹ Gathered using PSINFO (www.sysinternals.com)

Output from 'df -ah' command

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/hda3	32G	20G	11G	65%	/
none	0	0	0	-	/proc
usbdevfs	0	0	0	-	/proc/bus/usb
/dev/hda1	101M	9.2M	86M	10%	/boot
none	0	0	0	-	/dev/pts
none	504M	0	504M	0%	/dev/shm



Appendix D: References Used (in order of appearance)

1. SANS - Unknown Binary Media – taken from (http://www.giac.org/gcfa/binary_v1_4.zip)
2. Canadian Evidentiary Act. (<http://laws.justice.gc.ca/en/C-5/>)
3. Netcat Reference Material and RPM – (<http://rpmfind.rediris.es/rpm2html/redhat-8.0-i386/nc-1.10-16.i386.html>)
4. XMMS RPM's for Redhat – (<http://havardk.xmms.org/dist/xmms-1.2.7-rh8-rh9-rpm/>)
5. Google (www.google.ca) - BMAP Search Result (<http://lwn.net/2000/0420/announce.php3>)
6. Google (www.google.ca) 'bmap 1.0.20' Search Result (<http://build.lnx-bbc.org/packages/fs/bmap.html>)
7. DMZ Definition - (<http://isp.webopedia.com/TERM/D/DMZ.html>)
8. National Institute of Standards and Technology – Definition of Forensic Image (<http://www.cftt.nist.gov/DI-spec-3-1-6.doc>)
9. Canadian Evidence Act – (<http://laws.justice.gc.ca/en/c-5/15821.html>),
10. Tool Reference - PSINFO (www.sysinternals.com)
11. Michael Ford - Imaging of Volatile Memory using a Unix machine (<http://www.samag.com/documents/s=9053/sam0403e/0403e.htm>)
12. Rob Lee - SANS GCFA Training – in class notes on MAC times.